

Detecting Malware in Application Stores with Permission Trained Machine Learning

**Capstone Project
Fall 2021**

Zachary Honderich	Darshil Patel
108600180	109222174

School of Information & Communications Technology
Seneca College of Applied Arts and Technology
Toronto, Canada

ABSTRACT

With an estimate 3.5 billion users of smart phones (techjury), the risk of malware infecting the phone and its user is higher than ever, and of total smart phone users over 2.8 billion of them are users of Android smartphones (BusinessofApps). Due to the sheer size of users and the amount of malware they are susceptible to, we wanted to create a way for users to be able to recognize between a harmless application and a malicious one. As applications can only function based on the permissions they are given, we focused on how we can different malware from benign applications based on the permissions the application asks for. As a lot malicious applications function similarly to one another, they also have similar permissions they ask the user for. We want users to have a way of quickly and easily finding out whether an application they want to download is malware or not. We will do this by creating a trained machine learning model that will test against different applications and compare permissions of benign and malicious applications. With this testing our model will be able to differentiate between benign and malicious applications on an application store before the end user is ever required to download the application.

TABLE OF CONTENTS

TABLE OF CONTENTS	iii
1. Introduction.....	1
2. Background	3
2.1 Malware Within the Android Ecosystem.....	6
3. Architecture.....	8
4. Implementation	10
4.1 Training the Machine Learning Model	11
4.2 Permission Analysis and Display	13
5. EXPERIMENTS	15
5.1 Analysis	15
5.2 Areas for Future Expansion	17
6. CONCLUSION	18
References.....	19

LIST OF FIGURES

Figure 1. Diagram of the environment and user process	8
Figure 2. Amazon application being tested on webpage	10
Figure 3. Creation of permission maps into a CSV file.....	12
Figure 4. First 11 permissions in each permission maps of the first 17 applications	13
Figure 5. Types of classification available with their numerical value	14
Figure 6. Amazon application results displayed on webpage.....	14
Figure 7. The Accuracy of our model when analyzing dataset-2-04.....	16

1. INTRODUCTION

Malware is one of the fastest growing and ever-present risks for anyone who uses technology. Malware can come in many different forms, in many different ways and on different devices. Malware is an incredibly dangerous threat especially to Android devices, because it is so easy for a regular user to fall prey to malware. One of the main causes of malware on Android is through applications downloaded from various APK distribution store (Sufatrio et al.). Malicious applications can be put on the application store and make it seem like any other application, however their purpose and function is malicious. As the user cannot tell whether an application is malware or not, they are downloading the application with no sense of the possible danger. When downloading applications, users do not think of the risks that may be associated with the application. When an application is downloaded it will ask the user accept its permissions, and this is where the user gets caught. If the user agrees to the application's request for permissions, they allow the application to essentially do whatever it wants on the device; only limited by the scope of permissions that it was granted.

While there are certain applications on the application store that help users protect their device by scanning it to see any anomalies, it does not inherently reduce the risk or impact of the malware if it has already infected your device during the download or when allocating the application permissions. When malware is unleashed on your device, it can cause a lot of damage, such as stealing personal information, altering stored information, and serving ads on the device. The problem that we are trying to solve is to make users more aware that not every application is safe to download and give them a risk-free solution to check the applications intentions.

Most solutions that are implemented just check if there is malware on the Android devices, and this process relies on the suspect applications themselves being downloaded to the phone. These anti-malware applications scan your device and look for any threats, once threats are found will remove the malicious applications. Other solutions focus on scanning the application itself while it is being downloaded. While these solutions work and have a purpose, they still do not protect the device from installing malicious applications proactively while relying cleaning up the device after the fact.

Our solution will differ from the rest, as our solution does not require the user to interact with the application in any way. As some applications can infect devices while they are being installed (Kaspersky), our solution removes that risk entirely. The purpose of our solution is to inform the user whether the application they want to download is malicious or not, with the focus of our solution being to look at the permissions of the application. As users can view what permissions the application requires on the application store, there is no need to download the application to the user's device and see the permissions that way. Due to this we have removed the risk of having to download the application to the user's device, as the user interacts with our solution through a website.

Our approach to the solution of the problem utilizes machine learning, as it allows us to use cutting edge and promising technology to go about creating our own solution in the way we want. While designing the solution we looked at how the user would know if the application is malicious or not and how we would test the application for its maliciousness. These questions led us to creating our own unique solution that is both easy to use and is functional. Due to all of these factors our solution is user-friendly, easy-to-use and risk free.

2. BACKGROUND

Due to the nature of machine learning and its growth over the past years, it has been a powerful tool to help people increase their knowledge and skills in the security field. In particular machine learning can be very prominently used in the network security field. Machine learning can be used to block many kinds of malware attacks such as SYN flood attacks, smurfing and DDoS attacks (Aledhari et al. 2). Since machine learning is capable of doing this, it can also detect malware on the Android application store by utilizing permissions.

When analyzing malware on Android the machine learning techniques can be broadly put into either of the two groups, which are static analysis and dynamic analysis (Milosevic et al. 267). With the ever-growing increase of mobile devices using Android (Tam et al.), detecting malware on them is becoming a bigger and bigger issue, especially since developers can attempt release a malicious application that has malware loaded onto an application store. When users download applications, they do not know if they are downloading a legitimate application or a malicious application with malware and must trust the security controls put in place by the application repository they are using. Mobile malware can be categorized into three classes or types, they are: malicious applications with malware that steal the user's personal information or track the user, use the user's phone as a station to execute a distributed DoS attack, and toll-fraud attacks which exist to spam the user with advertises or steal their money (Islam et al. 92).

For static analysis, there can be two approaches for Android malware classification, they are permission-based analysis and source code-based analysis (Milosevic et al. 268). Permission-based analysis involves the use of permission names in order to create a model for machine learning, as most applications require applications from the user in order for the application to properly function. Source code-based analysis requires people to physically look through the

source code of an application to determine if it is malicious or not. One way to tell from viewing the source code of a malicious application is that a malicious application will usually use a combination of methods, API calls, or services in a different way than a harmless application (Milosevic et al. 268).

A more dynamic approach is looking at the behavior of an application when it is being ran or executed. To get a better understanding of the application and whether or not it is malware, multiple things have to be looked at, such as the interaction between the application and the Android mobile OS, behavioral patterns from API call event logs and the system must look at every behavior vector and judge it to decide if the application is acting like malware or not (Islam et al. 93).

An approach in malware detection is the use of API monitoring to inspect what an Android application may be doing, with many different approaches for this having already been published our group was able to determine aspects that worked well and others that would lead us toward a dead end. Aafer et al were able to come up with a very robust approach calling it DroidAPIMiner where they conduct analysis of Android APK files, and specifically look at which API calls are being made, from there they look at common calls between malwares.

Once common calls are established, they are able to use these to make very accurate predictions on if an APK is malware or not. A similar approach utilizing machine learning was taken by Wu et al. where they use the DroidAPIMiner as their comparison benchmark to determine how successful they are, their approach involved using a weighted system to determine how strongly each API is correlated with malicious activity and was found to have a very similar effectiveness as the team being DroidAPIMiner's solution. DroidAPIMiner was then used to support Zhang et al. and their paper on using API calls and feature exploration in conjunction with

machine learning to create malware detection methods that display an even higher degree of accuracy than just DroidAPIMiner's approach. Both papers have high degree of accuracy in the lab setting however the approach taken by Zhang et al. theoretically should have a higher amount of trust as it joins to very promising approaches together and is not reliant on a single group of data points to make its inferences.

A study by the Institution of electrical engineers into using ensemble learning which is the practice of using multiple machines learning outputs and combining the results together to form an inference was able to achieve a 97-99% correct detection rate in its tests using multiple different factors and many different machine learning algorithms to make an inference on if an application was malware or not. Mahindru et al. created FSdroid which is another machine learning application that looked into malware detection and found that it was feasible for malware detection to using a single data point and mentioned specifically that methods like DroidAPIMiner are viable even when only using one set of data.

In recent years, various approaches have to be devised to fight malware based on their nature. Some researchers have proposed their Android malware detection methods by using machine learning with the features such as operation codes (opcodes), API functions, system calls, system status, etc. using static analysis and dynamic analysis. Static analysis of malware does not require conditional, untrusted or sandboxed execution of malware once the original contents of the malware are visible, while dynamic analysis monitors system calls or system status see the chance of threat while the malware hides itself until there is a chance to attack, some of them prefer to use static analysis.

A paper Wu et al. proposed a system, called DroidMat which extracts the information such as permissions, API calls from the manifest file, then applies an algorithm that enhances the

malware modeling capability, and finally adopts another algorithm to classify the application as benign or malware. Another group made by M. Fan proposed to extract APIs from the .apk file to build a matrix according to the frequencies of their calling relationships, and then compare the matrix similarities between two .apk files. However, since the APIs are coarse-grained and sparse, it will be easily obfuscated by polymorphic techniques. (Sami et al)

As machine learning malware detection software processes, so does the malware it faces. Android systems to this day suffer from multiple attacks from malware applications because of its open platform, evidence shows that there are nearly 19 million cumulative samples of Android malicious applications (Wei et al. 25591).

The increasing number of malicious attacks will continue to threaten the security of an Android system and will threaten the security of private information of Android users. Therefore, ensuring evolution of machine learning malware detection would be in everyone's best interest. Those are the reasons why researchers are devoted to improving the efficiency and the effectiveness of new and the latest Machine Learning, in order to use them for malware detection.

2.1 Malware Within the Android Ecosystem

As Android is the most used mobile device platform, the threat of malware on it is much greater due to the freedom Android offers both developers and users (Senanayake et al.). As Android offers freedom to both developers and users, they are in full control of what they want to do on their device, which means they can create and download third-party applications and software. This creates a massive risk for users, as they do not know what is safe and what is not, but for malicious actors, this opens up a window of opportunity for malicious actors to compromise the user's device and steal personal information. According to research done by McAfee in 2014,

there are over six million samples of malware, and of that about 98% target Android devices. From the third quarter of 2014 to the fourth quarter there was a 14% rise (Tam et al.).

This is why researchers and developers have developed methods to counteract this threat. Malware detection on Android specific devices can be performed in two ways; they are signature-based and behavior-based methods (Senanayake et al.). There are also other who are using machine learning techniques on Android to have the edge over malware, some of these techniques are Deep Conventional Neural Network, Decision Trees, and Support Vector Machines (Senanayake et al.). Malicious actors do not just create their own malware, they also look for any vulnerabilities in existing applications, and so machine learning can also be used to detect any issues in the code of the application.

3. ARCHITECTURE

As our problem and solution focus on the permissions of applications that is what our model will have to focus on as well. The following figure shows the complete process of how the user interacts with the model and what the model does to find out whether the application is benign or malicious.

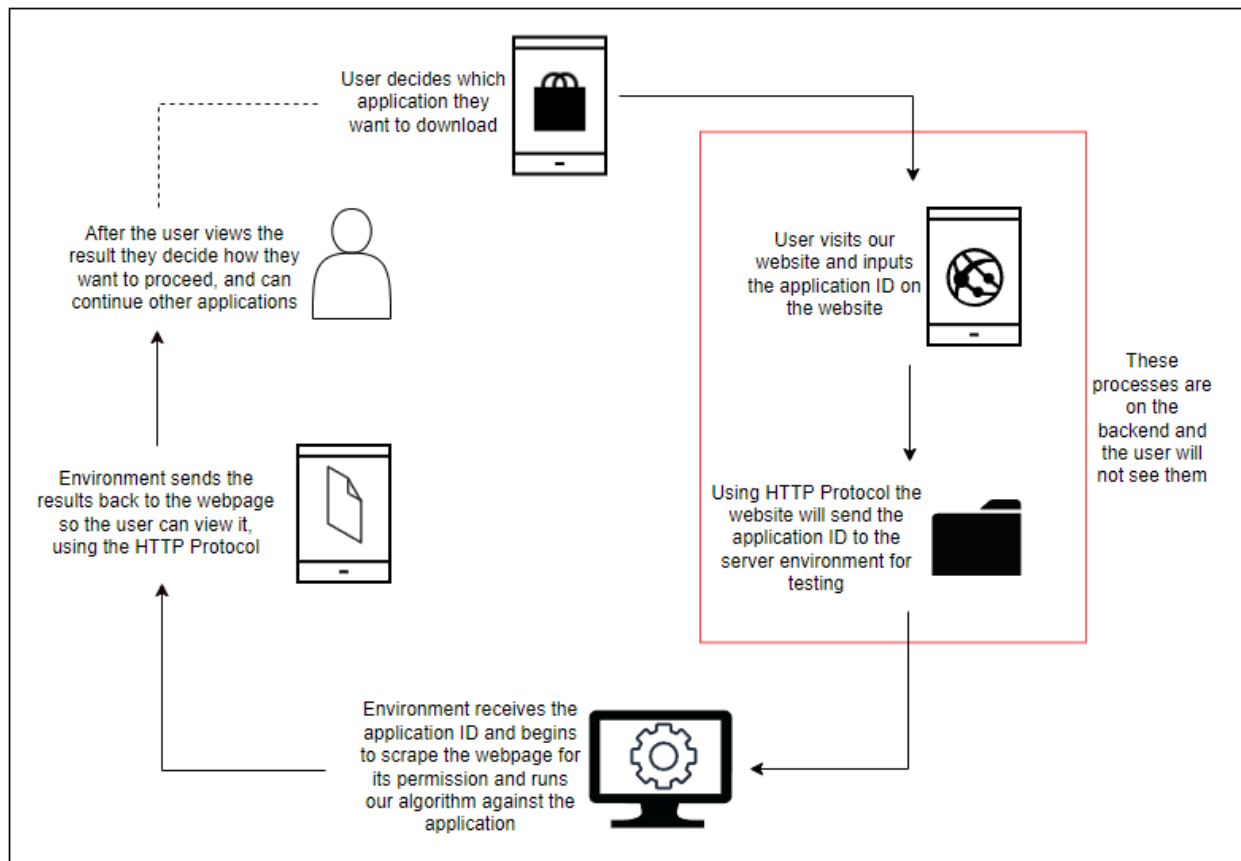


Figure 1. Diagram of the environment and user process

Our solution to using Android application permissions to determine if the application in question is malicious or benign has 3 primary steps to go from application to inference. To start, the user will identify an application that they would like to get an analysis of, the on an Android application store. A submission portal in which the application's link will be submitted consists of

an interactable text box and submission button which are accessible on a webserver hosted by a team running this malware detection solution either on physical servers, or within a cloud environment. The second step in phase one is acquisition of the APK file which will be obtained from the Android application store chosen by the user, this APK is downloaded into a temporary folder on the server which is secured and given limited permission as the application is an unknown file, that may contain malware. All aspects related to analysis including generated files, and auxiliary APKs downloaded with the suspect application will be contained in this folder that is fully deleted after analysis is complete.

Our solution uses a template that we have developed called a permission map, which is an arrangement of the indicators that specify the presence or lack thereof every possible Android permission within the suspect application. The permission analysis phase is where the permission map is tested against a trained machine learning model, this process is often completed in less than a second on all hosting options tested, this allows for a quick response time to the user. Once the prediction on the application is made a secondary results page to display the prediction is generated where the maliciousness of the application is displayed which could be either benign or the type of malware detected.

4. IMPLEMENTATION

The proof-of-concept solution we have developed works by displaying a Hyper-Text Markup Language (HTML) based webpage hosted on an Apache2 HTTP server (version 2.4.41). The landing page is a HTML document with an interactable text box, submit button and instructions on how to use the submission portal. The submission portal is required to be user friendly on both desktop and mobile based browsers so was developed with jQuery Mobile (Version 1.4.2). Once a link is inputted into the text box and the submission button is activated a python script is called with a GET request and the link is submitted for analysis.



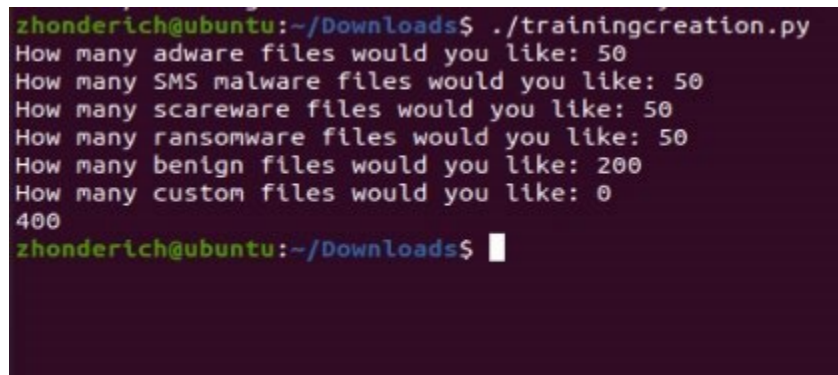
Figure 2. Amazon application being tested on webpage

The application store link is parsed for the application ID which is used as an identifier for downloading the applications APK file. Downloading the APK is accomplished with google-play-downloader [Rehmatworks], a free to use python script on available on GitHub. The APK file is download to a file with a randomly generated name as a UUID, for all practical purposes there will never have a repeat name as there are 2^{122} possibilities of UUIDs. All analysis, generation of support files, and auxiliary APK files will be contained within this file which has limited permissions to reduce risk to the environment running analysis. Once the application is successfully downloaded its permissions are extracted from the .APK file by utilizing Android Asset Packaging Tool (AAPT) package (version 1:8.1.0+r23) to output in the system level permissions into a text list. A permission map is created in which the list of permissions are converted into to a CSV file where all permissions are ordered in a single row. This row will have each permission being denoted by a “1” indicating the presence of a permission, and a “0” denoting the absence of a permission in the specific application. The permission map will be generated by comparing a list of all Android permissions in text form which is our master list of permission, then checking if each item in the master list appears within the text-based permission list for the application. The master list is laid out to check for each permission in a specific order, an example of this allows for “permission a” to always be the first column of the row no matter which application is being tested, “permission b” to be the second column and so-forth for all 324 permissions that can be granted by the Android operating system.

4.1 Training the Machine Learning Model

We used the Scikit-Learn (Version 1.0.1) machine learning library for Python to train our machine learning model, with Bernoulli Naive Bayes selected as the algorithm for training. To develop the datasets required to train our model we developed our own

tools/scripts over the course of project development, these tools include scripts to generate smaller datasets used for training and testing of the machine learning model. These sub-datasets are generated from two data sets CIC-AndMal2017 and CCCS-CIC-AndMal-2020 which together contain over 800 malicious and 800 benign application APK's. We developed tools to create small datasets of any ratio of malware to benign ware we could want which is outputted into a CSV file as multiple permission maps.

A terminal window with a dark purple background. The prompt is 'zhonderich@ubuntu:~/Downloads\$'. The user has run './trainingcreation.py'. The script asks for the number of files for each category: 'How many adware files would you like: 50', 'How many SMS malware files would you like: 50', 'How many scareware files would you like: 50', 'How many ransomware files would you like: 50', 'How many benign files would you like: 200', and 'How many custom files would you like: 0'. The script then outputs '400' and returns to the prompt 'zhonderich@ubuntu:~/Downloads\$' with a cursor.

```
zhonderich@ubuntu:~/Downloads$ ./trainingcreation.py
How many adware files would you like: 50
How many SMS malware files would you like: 50
How many scareware files would you like: 50
How many ransomware files would you like: 50
How many benign files would you like: 200
How many custom files would you like: 0
400
zhonderich@ubuntu:~/Downloads$
```

Figure 3. Creation of permission maps into a CSV file

The dataset creation tool works by generating a permission map for each application being added to the dataset, with to test the train our model we upload our dataset to our Google collab instance which is set up to train the new model from the collection of permission maps within the csv file. The dataset is laid out as a collection of permission maps within a single CSV file, in the specific case of this generated dataset there are 400 permission maps contained within it.

A1				0								
	A	B	C	D	E	F	G	H	I	J	K	L
1	0	0	0	0	0	1	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	1	0	0	0	0	0	
9	0	0	0	0	0	1	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	0	
16	0	0	0	0	0	1	0	0	0	0	0	
17	0	0	0	0	0	0	0	0	0	0	0	

Figure 4. First 11 permissions in each permission maps of the first 17 applications

Initially in the dataset each application has its correct classification numerical value added to the end of its respective permission map which for our supervised training to test the accuracy during the training process.

4.2 Permission Analysis and Display

To begin the analysis of the application we import our trained machine learning model into a python script with pickle python package. We will use the model to predict the classification of an application based on its permission map inputted permission map, this typically takes less than one second to complete and does not add a significant amount of time for the user to wait during the analysis phase. Once the prediction has been made about the model, the prediction is outputted as the numerical value of its classification which we translate the classification number to a human readable tag.

Adware	0
SMS Malware	1
Scareware	2
Ransomware	3
Benign	4

Figure 5. Types of classification available with their numerical value

Finally, to display the prediction to the user end user we generate a results page with Python3 which is, like the submission page, is designed to work well on Desktop and Mobile, and again was designed with jQuery Mobile (Version 1.4.2) to meet usability and aesthetic expectations.

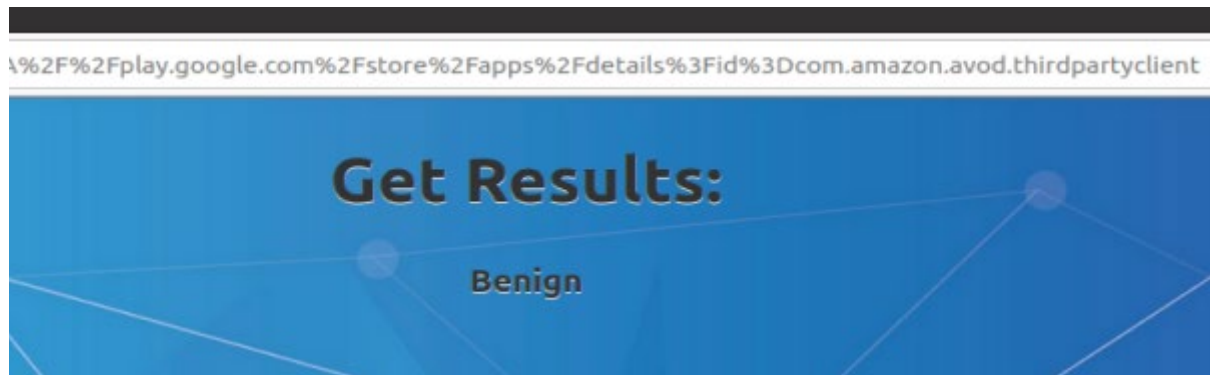


Figure 6. Amazon application results displayed on webpage

5. EXPERIMENTS

Now that our environment has been created it was time to test it to make sure it is functioning correctly and ensure all of the individual components work together seamlessly. For our testing, we tested both of our dataset APKs and as well as the APKs of actual application that are found on the application store. We tested application store applications because we were able to test the workflow of our entire solution and ensure that there were no issues with different components interacting. As our datasets contain randomly selected APKs from a list, we tested our model with a few different datasets to get a broader, and more accurate averaged result across many tests. To start, we began testing real applications to get a baseline on how well our model would perform.

5.1 Analysis

To validate our results, we reused our tool to create sub-datasets for training and instead used it to make datasets with different ratios of malicious to benign applications for testing accuracy, which would allow us to do multiple analysis rounds to get an average of how often we had a correct prediction. First, we upload our dataset to our Google collab instance which is set up to analyze all permission maps within the csv file.

Initially in the dataset each application has its correct classification numerical value added to the end of its respective permission map which can be used to validate if the prediction made by the model is correct. In conjunction the dataset creation tool we created a tool for testing the accuracy of the trained model that can be used to validate the predictions for every application quickly. The classification number is taken from the final column of each row in the permission map collection CSV and appended to a master

classifications CSV which will be compared to the output CSV of predictions script after predictions are made.

To get a prediction for every permission map within the dataset we loop through all individual permission maps and append our numerical prediction value into single row contained in the aforementioned CSV file in the same order that the correct values were appended to the master classifications CSV. Finally, both CSVs are imported as lists into Python as lists and compared to find matching predictions with respect to the correct classification for each item.

The output of the accuracy file is the numerical value of the inference made by the model which could be 0 through 4. The numerical inference is then added to a CSV file which contains a prediction for every application being tested in a single row of a csv file. Finally, the predictions CSV is compared to a master list of correct classifications for each application in the sub-dataset and the results are displayed as a percentage.

```
zhonderich@ubuntu:~/Downloads$ ./mapcomp.py
 0  1  2  3  4  5  6  7  8
399
0  0  0  0  0  0  0  0  0
 4

[1 rows x 400 columns]
 0  1  2  3  4  5  6  7  8
399
0  0  4  4  4  3  0  2  0
 4

[1 rows x 400 columns]

Accuracy:
0.6825
```

Figure 7. The Accuracy of our model when analyzing dataset-2-04

5.2 Areas for Future Expansion

With a working model and environment, it was time to see on what areas we could improve upon. Our model has a nearly 70% accuracy rate, and so we wanted to try and increase that number as much as possible. From our results we get a baseline of how our model is performing and from there we can parameter tune to try and get it to be more efficient and more accurate. Since our environment is working and all of the components are integrated together, the main areas of improvement are adding security protocols to our webpage, improving our algorithm and perhaps also adding further test cases and depending on how they perform we continue to make tweaks.

We have also found that some applications that are legitimate such as the Uber application will be shown as malware if tested. For applications like these that are proven benign and are trusted in industry a whitelist may be included in future releases that will skip scanning applications that are known to be benign but could register as malicious.

6. CONCLUSION

Due to the increase in risk and effect of malware it is becoming more and more important in order to protect ourselves against it, in every way possible. Our solution hopes to bring a reliable and simple to use product that users can utilize so as to not put their mobile device and their personal information at risk. As more and more people create applications and put them on the application store, it becomes all the more difficult to differentiate between the actual applications and malware. As applications need permissions in order to function, our model will test against them and make a clear and concise basis on the application's maliciousness and inform the user, who will then decide on what course of action to take. They can either continue to download and use the application or not download it. Either way, our model will provide a good base for the users to make a judgement, but the ultimate decision is up to them.

REFERENCES

- Aledhari, Mohammed, Rehma Razzak, and Reza M. Parizi. "Machine Learning for Network Application Security: Empirical Evaluation and Optimization." *Computers & Electrical Engineering* 91 (2021): 107052. <https://doi.org/10.1016/j.compeleceng.2021.107052>.
- Faruki, Parvez, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. "Android Security: A Survey of Issues, Malware Penetration, and Defenses." *IEEE Communications Surveys & Tutorials* 17, no. 2 (2015): 998–1022. <https://doi.org/10.1109/comst.2014.2386139>.
- Garg, Shivi, and Niyati Baliyan. "Data on Vulnerability Detection in Android." *Data in Brief* 22 (2019): 1081–87. <https://doi.org/10.1016/j.dib.2018.12.038>.
- Giacinto, Giorgio, and Belur V. Dasarathy. "Machine Learning for Computer Security: A Guide to Prospective Authors." *Information Fusion* 12, no. 3 (2011): 238–39. <https://doi.org/10.1016/j.inffus.2011.02.001>.
- Islam, Nayeem, Saumitra Das, and Yin Chen. "On-Device Mobile Phone Security Exploits Machine Learning." *IEEE Pervasive Computing* 16, no. 2 (2017): 92–96. <https://doi.org/10.1109/mprv.2017.26>.
- Martín, Ignacio, José Alberto Hernández, and Sergio de los Santos. "Machine-Learning Based Analysis and Classification of Android Malware Signatures." *Future Generation Computer Systems* 97 (2019): 295–305. <https://doi.org/10.1016/j.future.2019.03.006>.
- Milosevic, Nikola, Ali Dehghantanha, and Kim-Kwang Raymond Choo. "Machine Learning Aided Android Malware Classification." *Computers & Electrical Engineering* 61 (2017): 266–74. <https://doi.org/10.1016/j.compeleceng.2017.02.013>.

- Suarez-Tangil, Guillermo, Juan E. Tapiador, Pedro Peris-Lopez, and Jorge Blasco. "Dendroid: A Text Mining Approach to Analyzing and Classifying Code Structures in Android Malware Families." *Expert Systems with Applications* 41, no. 4 (2014): 1104–17. <https://doi.org/10.1016/j.eswa.2013.07.106>.
- Wei, Linfeng, Weiqi Luo, Jian Weng, Yanjun Zhong, Xiaoqian Zhang, and Zheng Yan. "Machine Learning-Based Malicious Application Detection of Android." *IEEE Access* 5 (2017): 25591–601. <https://doi.org/10.1109/access.2017.2771470>.
- Wu, Qing, Xueling Zhu, and Bo Liu. "A Survey of Android Malware Static Detection Technology Based on Machine Learning." *Mobile Information Systems* 2021 (2021): 1–18. <https://doi.org/10.1155/2021/8896013>.
- Wu, Songyang, Pan Wang, Xun Li, and Yong Zhang. "Effective Detection of Android Malware Based on the Usage of Data Flow Apis and Machine Learning." *Information and Software Technology* 75 (2016): 17–25. <https://doi.org/10.1016/j.infsof.2016.03.004>.
- Yerima, Suleiman Y., Sakir Sezer, and Igor Muttik. "High Accuracy Android Malware Detection Using Ensemble Learning." *IET Information Security* 9, no. 6 (2015): 313–20. <https://doi.org/10.1049/iet-ifs.2014.0099>.
- Zhang, Jixin, Zheng Qin, Kehuan Zhang, Hui Yin, and Jingfu Zou. "Dalvik Opcode Graph Based Android Malware Variants Detection Using Global Topology Features." *IEEE Access* 6 (2018): 51964–74. <https://doi.org/10.1109/access.2018.2870534>.
- Zhang, Nan, Yu-an Tan, Chen Yang, and Yuanzhang Li. "Deep Learning Feature Exploration for Android Malware Detection." *Applied Soft Computing* 102 (2021): 107069. <https://doi.org/10.1016/j.asoc.2020.107069>.

Mahindru, Arvind, and A.I. Sangal. "FSDroid:- A Feature Selection Technique to Detect Malware from Android Using Machine Learning Techniques." *Multimedia Tools and Applications* 80, no. 9 (2021): 13271-3323. doi:10.1007/s11042-020-10367-w.

Aafer, Yousra, Wenliang Du, and Heng Yin. "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android." *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering Security and Privacy in Communication Networks*, 2013, 86-103. doi:10.1007/978-3-319-04283-1_6.

"Search UNB." University of New Brunswick Est.1785. Accessed November 23, 2021.

Tiwari, Shashwat. "Android Malware Dataset for Machine Learning." Kaggle. March 13, 2021. Accessed November 23, 2021.

Ray, Sunil. "Commonly Used Machine Learning Algorithms: Data Science." Analytics Vidhya. August 26, 2021. Accessed November 23, 2021.

"Google-play-scraper." PyPI. Accessed November 23, 2021.

Google APIs Terms of Service | Google Developers. Accessed November 23, 2021.

S, Surabhi. "Beginners Guide to Naive Bayes Algorithm in Python." Analytics Vidhya. January 16, 2021. Accessed November 23, 2021.

"Learn." Scikit. Accessed November 23, 2021.

Rehmatworks. "Rehmatworks/gplaydl: Command Line Google Play APK Downloader. Download APK Files to Your PC Directly from Google Application store." GitHub. Accessed November 23, 2021.

Sami, A., Yadegari, B., Peiravian, N., Hashemi, S., & Hamze, A. (2010). Malware detection based on mining API calls. *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*. <https://doi.org/10.1145/1774088.1774303>

Sufatrio, Darell J. Tan, Tong-Wei Chua, and Vrizlynn L. Thing. "Securing Android." *ACM Computing Surveys* 47, no. 4 (2015): 1–45. <https://doi.org/10.1145/2733306>.

Tam, K., Feizollah, A., Anuar, N. B., Salleh, R., & Cavallaro, L. (2017). The evolution of Android malware and Android Analysis Techniques. *ACM Computing Surveys*, 49(4), 1–41. <https://doi.org/10.1145/3017427>

Senanayake, Janaka, Harsha Kalutarage, and Mhd Omar Al-Kadri. "Android Mobile Malware Detection Using Machine Learning: A Systematic Review." *Electronics* 10, no. 13 (2021): 1606. <https://doi.org/10.3390/electronics10131606>.

G, Deyan. "67 Revealing Statistics about Smartphone Usage in 2021." TechJury. December 07, 2021. Accessed December 11, 2021.

Curry, David. "Android Statistics (2021)." Business of Apps. November 15, 2021. Accessed December 11, 2021.

"20 Scary Mobile Malware Statistics to Keep in Mind." Safe at Last. August 11, 2021. Accessed December 11, 2021.

"What Is a Drive by Download." www.kaspersky.com. January 13, 2021. Accessed December 11, 2021.